Table of Contents

Prerequisites for integrating Calldorado Caller SDK

Before integrating our Caller SDK you must have an account created on https://my.calldorado.com/. On the website is a quick integration guide that you need to follow to get two important ID's, namely the Account ID and an application ID. These two IDs must be added to the host app as part of the integration. More details will follow.

1.0 Basic integration of Caller SDK

This chapter describes the very basic of getting the SDK integrated and running. You will be taken through the below steps for the integration

- Requirement for your development environment and how to set it up
- Android Manifest entries and necessary permissions
- Gradle changes
- How and when to initialize

1.1 Requirements

- Turn off 'instant run' in Android Studio while building debug and CDO is activated (File/Settings/Build, Execution)
- **Set targetSdk to 29**
- Upgrade local versions of libraries to match imported Google code as needed. FX Standard Firebase (Core) needs to be 16.0.5 to match Google 17.1.1

Below table shows environment settings, which we have used in test apps for the most recent Caller SDK versions. In some cases, these settings don't have to be exact.

Environment	**CDO 6.0**
Android Studio	4.0+
Gradle version (wrapper-properties)	5.4.1
Android Gradle plugin version (project gradle)	3.5.1
JAVA JDK	1.8.0
minSdkVersion	17
compileSdkVersion	30

Table 1 - Environment requirements for recent CDO versions

1.2 AndroidManifest entries

From the beginning of 2019, read_call_log permission will require a special permission from Google to use. Not having this permission will disable Caller SDK ability to read the phone number on Pie+ devices, but we will show call screens with limited information for those devices.

If you do not plan to request this permission from Google, add following entry in your Manifest to remove the permission:

```
<uses-permission android:name="android.permission.READ_CALL_LOG"
tools:node="remove"/>
```

Under the <application> tag, add the Caller SDK account/app IDs as meta tags as follows:

```
<meta-data android:name="com.calldorado.AccountId" android:value="YOUR ID"/>
<meta-data android:name="com.calldorado.AppId" android:value="YOUR ID"/>
```

1.3 Gradle (app) entries

Enable Multidex if you haven't already, use an online guide if you are in doubt of how to.

1.3.1 Repositories

The Caller SDK operate with two levels of builds: Snapshots and Releases. Release will always be well tested whereas a snapshot may contain untested or specialized code. Only use a snapshot version if told so by a Calldorado employee and if so, make sure to use the exact same version name. Avoid using '+' for version as it can lead to unexpected behavior. If not using snapshot, you can omit that repo in above Gradle entry, otherwise you need to add username/password to the entry.

Personal(snapshot) repository

If you have a personal repository add below repository tag to you gradle(app) file):

```
repositories {
    mavenCentral()
    maven {
        credentials {
            username = 'Your repo username'
            password = 'Your repo password'
        }
        url 'http://maven.calldorado.com/nexus/content/repositories/customer/'
    }
    maven { url 'http://maven.calldorado.com/nexus/content/repositories/thirdparty/' }
  }
```

Release repository

If you are using the release repository add below repository tag to you gradle(app) file):

```
repositories {
    mavenCentral()
    maven { url 'http://maven.calldorado.com/nexus/content/repositories/thirdparty/' }
    maven { url 'http://maven.calldorado.com/nexus/content/repositories/releases' }
}
```

1.3.2 implement the SDK

Under dependencies add:

```
implementation('com.calldorado:calldorado-release:6.0.xxxx@aar') {
    transitive = true
}
```

or if using personnel repository(snapshot):

```
implementation('com.calldorado:calldorado-release:6.0.XX.XXXX-SNAPSHOT@aar') {
    transitive = true
}
```

1.4 Optin

To be Google compliant it is required to show an Optin providing information to users about the Caller SDK functionality before we may activate the SDK.

1.4.1 Creating custom optin

When creating your own optin we need a couple of things provided from your side. First of you need to ask for the permissions required for the SDK to work. Ask for:

- CALL_PHONE
- READ_PHONE_STATE
- READ_CONTACTS
- Overlay permission(Optional on Android version 9 and below but mandatory for Android version 10 and above)

SDK will work with only CALL_PHONE and READ_PHONE_STATE but will perform better with the READ_CONTACTS permission.

Secondly we will need to know if the EULA consent has been given by the end user. This will be done on the EULA screen provided by you “Key Account manager” if you have not received any suggestions on how to design the EULA screen contact your “Key Account manager”. When the user accepts you’ll need to paste in and call this method:

```
private void eulaAccepted(){
HashMap<Calldorado.Condition, Boolean> conditionsMap = new HashMap<>();
conditionsMap.put(Calldorado.Condition.EULA, true);
conditionsMap.put(Calldorado.Condition.PRIVACY_POLICY, true);
Calldorado.acceptConditions(this, conditionsMap);
}
```

If the end user declines then do not call above mentioned method. More information about EULA can be found in section 4.0.

1.4.2 OPT-IN

> **Note: We recommend you to create your own OPT-IN. If you have any doubts regarding OPT-IN then please contact your KAM(Key Account Manager) at Calldorado**

All end users must accept the end-user license agreement and the privacy policy for both the app and the SDK to be allowed to run. This is referred to as OPT-IN. After the opt-in, the app will be fully operational and the SDK will be working sending/receiving data. A user can at any point opt-out using the Calldorado settings or via an in-app method. This is all needs to be Google compliant. The OPT-IN must provide information to users about the Caller SDK functionality before we may fully activate the SDK.

Please follow the section “**6.0 Using Your Own Opt-In** ” in the document to create your own opt-in to

onboard users.

1.5 Stable Activity entries

By stable Activity we mean an Activity, which not only exist for a short moment like a Splash Activity. Caller SDK initialization requires server communication and hence some time to run. We need the context fed to the initialization method to not get destroyed while the initialization is taking place, or the initialization may fail until initialization is called again.

Add the method to initialize The Caller SDK in typically the onCreate() method as follows:

```
Calldorado.startCalldorado(this);
```

Above method is available in different versions, which may provide feedback on when initialization is done or how the user responded to Optin and permissions using a callback. Optimally code to customize Aftercall colors and provide a better user experience if navigating from Aftercall to host app should be added to this Activity as well, see more in chapter 2.

1.6 Test Caller SDK functionality

Having done steps 1.0 to 1.5 you should now be ready to test that Caller SDK is operating properly. Start your host app to trigger the CDO initialize method. Make a phone call to or from a phone with network and check that you see a WIC (caller ID) during the call and then an Aftercall screen after the call.

2.0 Optimal Methods to change the look of the Aftercall

2.1 How to customize the colors, icons and look for the Caller SDK 5.8+

The look and feel of the WIC, the Aftercall and the building dialogs can be tailored in many ways with respect to aligning it to the host/publisher app. Two examples are given below.

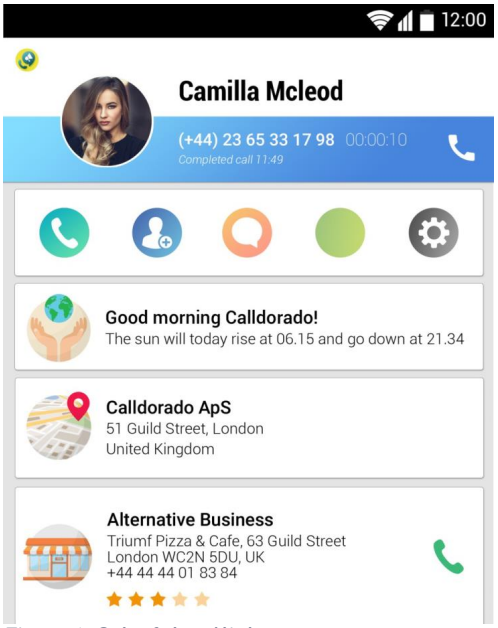


Figure 1, Colorful and light

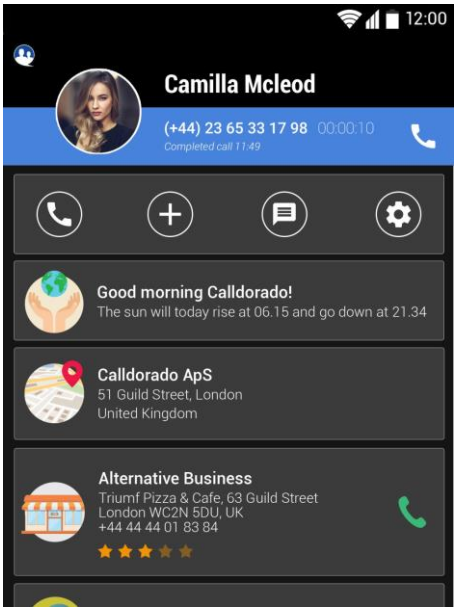
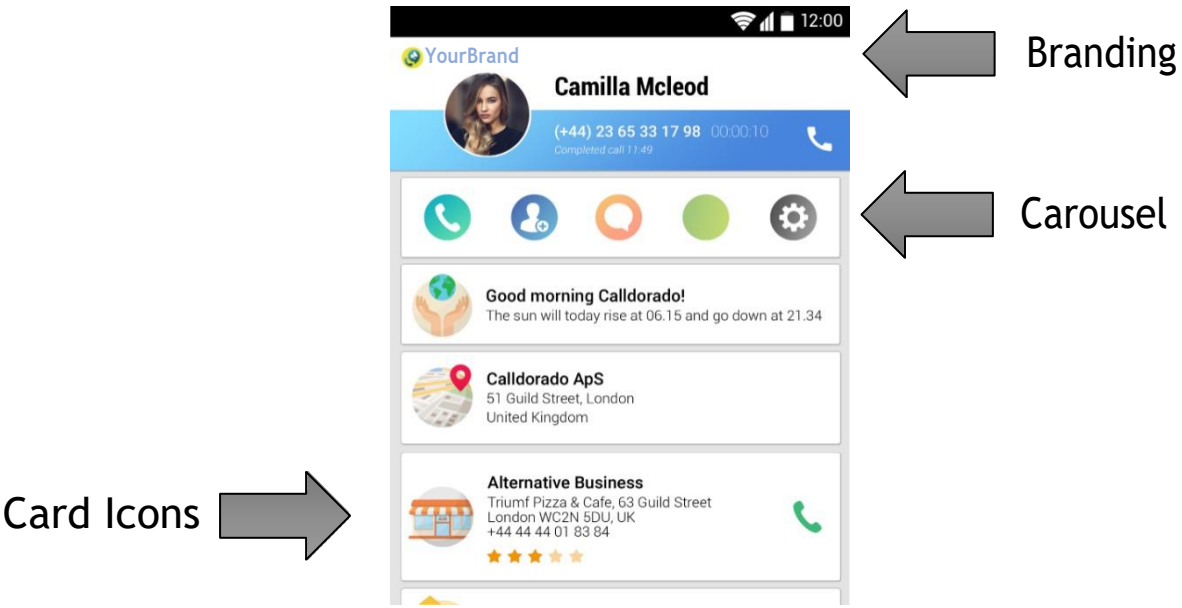


Figure 2, Dark theme and icons made more stylistic

How to customize the Aftercall

There are four areas that can be changed on the Aftercall. The icons shown in the carrousel can all be substituted with another icon. Each card has a default icon. It can either be changed or turned off. In both cases if not set, the default icons will be used.

In the contact card at the top, an area is reserved for the host app to set a logo or a like to make the aftercall more branded to the host app.



How to customize the Aftercall colors

Beside the icons and branding, it is possible to customize colors of the Aftercall: cards, background, font, circle around the contact image, background of the contact image and more.

The methods available are:

```
Calldorado.setCalldoradoCustomColors(this, <hashmap>); // set new colors
Calldorado.resetCalldoradoColors(this); // set all colors to default
```

The colors can be changed dynamically on the fly. Depending on time of the year or the state of the app, the aftercall can be changed as desired.

The name of the elements that can be changed is listed below and explained in more details:

```
public enum ColorElement {
     AftercallBgColor, // background under the cards
     AftercallStatusBarColor, // action bar -do not set too light
     AftercallAdSeparatorColor, // ad separator
     CardBgColor, // card background
     CardTextColor, // card text
     CardSecondaryColor, // icon background for cards in Settings
     InfoTopTextIconColor, // text from top of caller info box
     InfoTopBgColor, // background from top of caller info box
     InfoBottomTextIconColor, // text & icon from bottom of caller info box
     InfoBottomRightBgColor, // right side from bottom of caller info box
     InfoBottomLeftBgColor, // left side from bottom of caller info box
     InfoCircleBorderColor, // border from caller image (AC & WIC)
     InfoCircleBgColor, // background from caller image (AC & WIC)
     InfoCircleImageColor, // icon or initials from caller image (AC & WIC)
     DialogBgColor, // dialogs background
     DialogHeaderTextColor, // dialogs header
     DialogSummaryTextColor, // dialogs text
     DialogButtonTextColor, // dialogs button
     WICTextAndIconColor, // WIC text and icons
     WICBgColor // WIC background color
 }
```

Example of the usage:

```
HashMap<Calldorado.ColorElement, Integer> colorMap = new HashMap<>();

colorMap.put(Calldorado.ColorElement.InfoBottomLeftBgColor, Color.LTGRAY);
colorMap.put(Calldorado.ColorElement.InfoBottomRightBgColor,
       Color.parseColor("#FFFFFF"));
colorMap.put(Calldorado.ColorElement.InfoBottomTextIconColor,
       getColor(R.color.colorPrimary));
```

```
Calldorado.setCalldoradoCustomColors(this, colorMap);
```

Colors not set will be using the default colors or previous set colors.

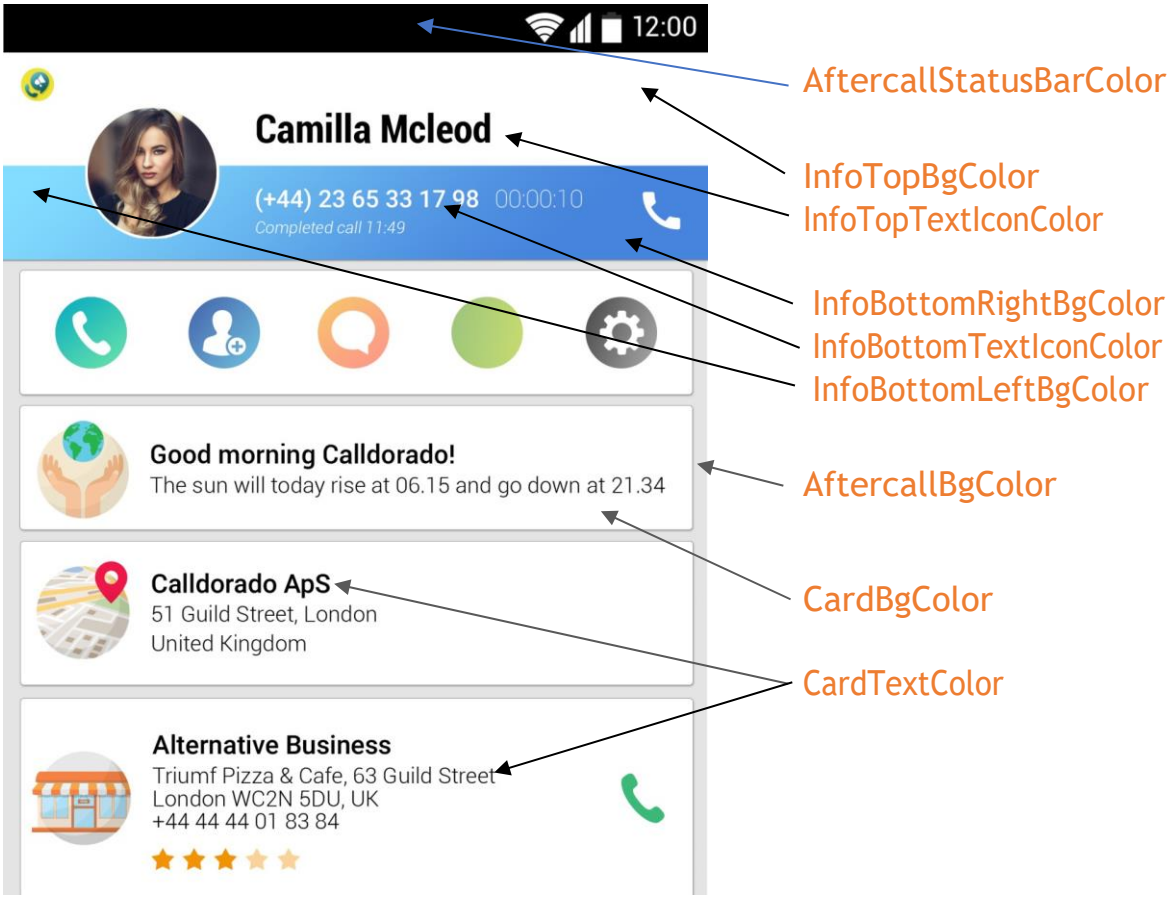


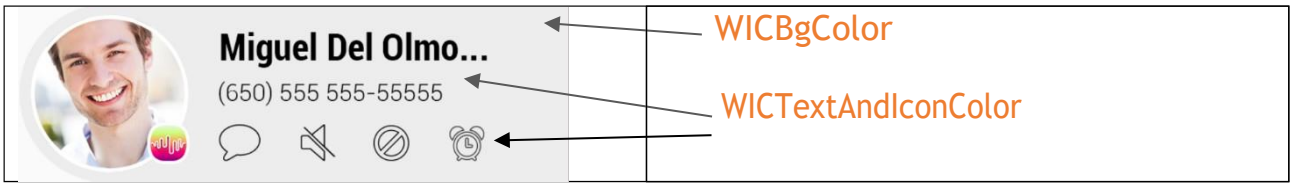
Figure 4, color elements

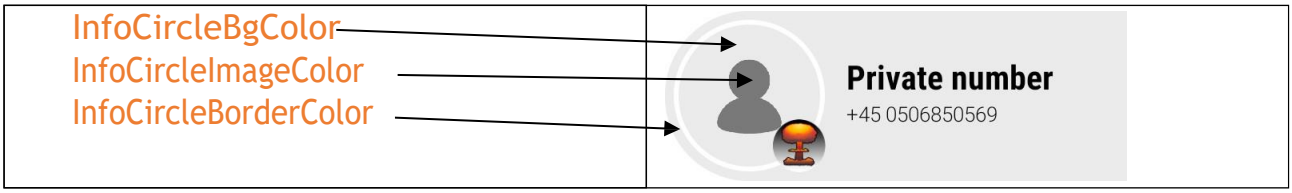
Just before the ad, there is a separator. Depending on the overall color scheme, the color of the separator may also need to be changed to make a clear border between the ad and the cards.

The key to be set for this is: AftercallAdSeparatorColor

How to customize the WIC

Using the same method as for the aftercall colors, setCalldoradoCustomColors(), the colors of the WIC can be changed.




InfoCircleBgColor
InfoCircleImageColor
InfoCircleBorderColor
Private number
+45 0506850569

How to customize the Aftercall icons

Beside the aftercall colors, it is possible to customize icons of the cards, carousel and branding area. The methods available are:

```
Calldorado.setCalldoradoCustomIcons(this, <hashmap>); // set new icons
Calldorado.resetCalldoradoIcons(this); // set all icons to default
Calldorado.removeAftercallCardIcons(this); // remove icons from all cards
Calldorado.resetAftercallCardIcons (this); // set card icons to default
Calldorado.setAftercallBrand(this, "resource_name"); // set brand image
```

The icons can be changed dynamically. Depending on time of the year or the state of the app, the aftercall can be changed as desired.

The name of the elements that can be changed are listed below and explained in more details:

```
public enum IconElement {
    ReEngagementCard, //show when setup from myCalldorado.com
    GreetingsCard, // shown when user location is known (sunrise/sunset info)
    SummaryCard, // shown if READ_CALL_LOG permission is granted
    AddressCard, // shown when caller location is known
    MissedCallCard, // shown if READ_CALL_LOG permission is granted
    WeatherCard, // shown when user location is known
    EmailCard, // shown when caller email is known
    HistoryCard, // always shown with historical info of the day
    RateBusinessCard, // shown when number is business
    HelpUsIdentifyCard, // not shown if number is from EEA
    SearchOnGoogleCard, // shown when number is unknown
    WarnYourFriendsCard, // shown when number is spam
    AlternativeBusinessCard, // shown when number is business
    CallAction, // call back
    SaveContactAction, // save contact if not in phonebook yet
    EditContactAction, // edit contact if already in phonebook
    MessageAction, // triggers system chooser with messaging options
    QuickMessageAction, // shown if SEND_SMS permission is present
    SettingsAction, // opens Settings activity
    BackToAftercallAction // shown when users clicks on the address card
}
```

Example of the usage:

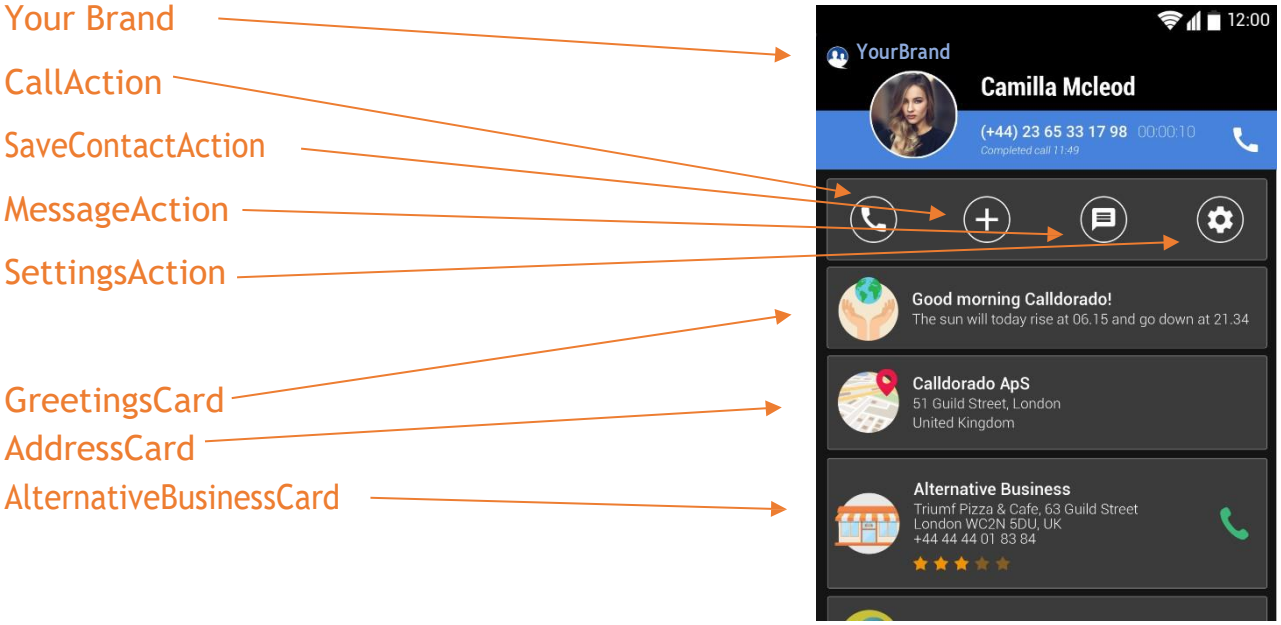
```
    HashMap<Calldorado.IconElement, String> iconMap = new HashMap<>();

    colorMap.put(Calldorado. IconElement.AddressCard, ""); // default
    colorMap.put(Calldorado. IconElement.GreetingsCard, null); // no icon
    colorMap.put(Calldorado. IconElement.CallAction, "my_icon_name");

    Calldorado.setCalldoradoCustomIcons(this, iconMap);
```

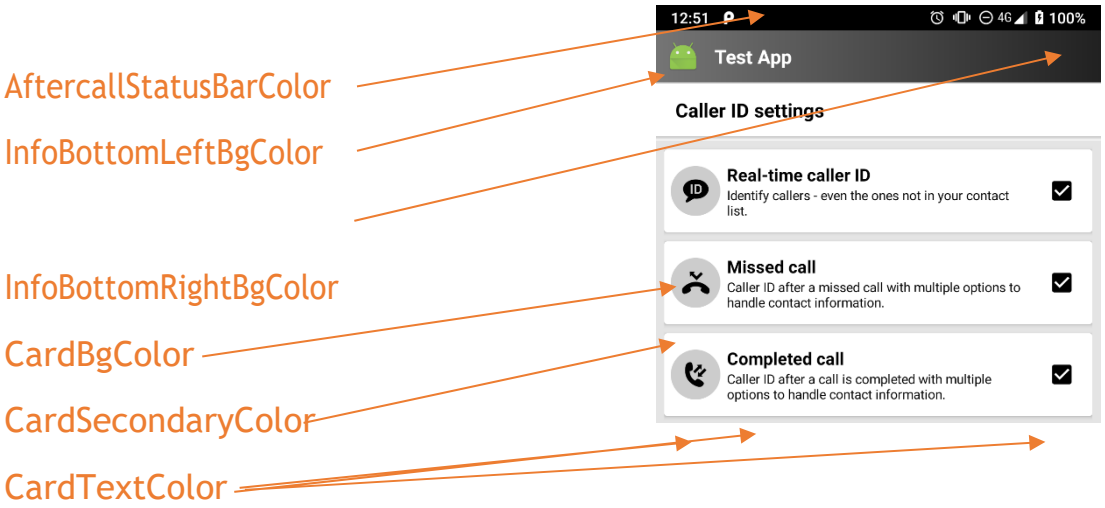
The icons must be in the drawable folders. We recommend that the new icons are minimized and then converted to different resolutions to prevent out of memory issues.

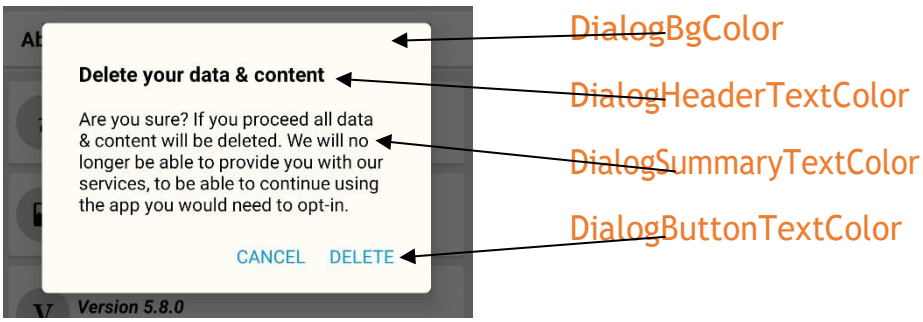
Icons not set will be using the default icons or previous set icons.



How to customize Settings

The colors of the Settings screen and dialogs will be changed together with the Aftercall colors.





2.2 Prevent Multiple Instances When Navigating from Aftercall to Host App

To improve Google compliancy by linking the host app and CDO closer together, we have added a link from Aftercall screen back to the host app on the top left corner app icon. The link should respect the stack of the host app if already present, but to prevent the possibility of adding launch Activity on top of the existing stack, you will need to add the following entry to the onCreate() of launch Activity (place just after super.onCreate(savedInstanceState);

```
if (!isTaskRoot() && getIntent() != null &&
getIntent().getBooleanExtra("onAppIconClick", false))

    Log.d("test", "finishing activity onCreate()");
    finish();
    return;
}
```

To tell the host app that it has been started or resumed from a CDO Aftercall, we send a Boolean extra with the intent. To set special behavior if the host app is started from CDO, place the following entry in the onCreate(..) of launch Activity:

```
if (getIntent() != null && getIntent().getBooleanExtra("onAppIconClick", false))
{ Your code }
```

2.3 New Option to Show Information of Latest Phone Call

Calldorado.showLastCallScreen(context); will show an Aftercall screen containing the information from the latest phone call of a user. Method will show a Toast if no call has yet happened since Caller SDK was installed.

2.4 Add custom UI to Caller SDK Aftercall

It is possible to add your own UI element to the Aftercall to enhance the user experience, to reengage the user's interest in your app and to tighten the bond between your app and the Caller SDK functionality. The latter is of big importance to Google when evaluating individual apps.

Setting up the native field to the Aftercall is simple. You tell us, that we should look for your native UI using a receiver, you provide the UI and handle its actions in your own code, and we will wrap your UI in a Card to be shown on each Aftercall. Adding your own native field requires following 3 steps:

Add following entry receiver to Manifest under <application> tag:

```
<receiver android:name="com.your.package.SetupFragmentReceiver" android:enabled="true">
  <intent-filter android:priority="101">
    <action android:name="android.intent.action.PHONE_STATE"/>
  </intent-filter>
</receiver>
```

Add following receiver:

```
package com.your.package.nativeac;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import com.calldorado.Calldorado;

public class SetupFragmentReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        if (intent.getAction().equals("android.intent.action.PHONE_STATE")) {
            Calldorado.setAftercallCustomView(new AftercallCustomView(context));
        }
    }
}
```

Finally add the class to handle the Native view. This class must contain a constructor and the method getRootView(). The example below uses a xml with an outer LinearLayout:

```
package com.your.package;

import android.content.Context;
import  android.util.Log;
import android.view.View;
import android.widget.LinearLayout;
import com.calldorado.android.ui.views.custom.CalldoradoCustomView;
import com.sappalodapps.callblocker.R;

public class AftercallCustomView extends CalldoradoCustomView {

    private LinearLayout ll;
    private Context context;

    public AftercallCustomView(Context context) {
        super(context);
        this.context = context;
    }
```

```
    @Override
    public View getRootView() {
        Log.d(TAG, "onCreateView() 1");
        ll = (LinearLayout) inflate(context, R.layout.aftercall_native_layout,
getLinearViewGroup());
        return ll;
    }
}
```

2.5 Advanced Caller SDK interface methods

Method returning a Settings Object, which can be used to show if a user has deactivated CDO through settings:

```
public static Setting getUserSettings(@NonNull Context mContext)
```

Method to set the colors of the WIC (Caller SDK UI):

```
public static void setCalldoradoWICColors(@NonNull Context mContext, int
textAndIconColor, int bgColor)
```

Method to set targeting data such as age or gender if any data available. Some ad providers will use this data and providing it could possible lead to better ad revenue:

```
HashMap<Calldorado.TargetingOption, String> targetMap = new HashMap<>();
targetMap.put(Calldorado.TargetingOption.Interests, "music,shopping");
targetMap.put(Calldorado.TargetingOption.BirthDate, "1992-01-30");
targetMap.put(Calldorado.TargetingOption.Gender, "male");

public static void setTargetingOptions(@NonNull Context context, @NonNull
HashMap<Calldorado.TargetingOption, String> devTargetingOptions)
```

Method which will return a Calldorado settings Activity to be shown fx in host app settings:

```
public static void createCalldoradoSettingsActivity(@NonNull Activity mActivity)
```

Method which can be used to change the user settings of Caller SDK. Avoid changing user settings already set by a user.

```
HashMap<Calldorado.SettingsToggle, Boolean> settingsMap = new HashMap<>();

settingsMap.put(Calldorado.SettingsToggle.REAL_TIME_CALLER_ID, true);
settingsMap.put(Calldorado.SettingsToggle.MISSED_CALL, true);
settingsMap.put(Calldorado.SettingsToggle.COMPLETED_CALL, true);
settingsMap.put(Calldorado.SettingsToggle.NO_ANSWER_CALL, true);
settingsMap.put(Calldorado.SettingsToggle.UNKNOWN_CALL, true);
settingsMap.put(Calldorado.SettingsToggle.CALLER_ID_FOR_CONTACTS, true);
settingsMap.put(Calldorado.SettingsToggle.LOCATION_ENABLED, true);
settingsMap.put(Calldorado.SettingsToggle.NOTIFICATION_REMINDERS, true);

Calldorado.setCalldoradoSettings(getMainActivity(), settingsMap);
```

Method which can be used to request Caller SDK and/or other permissions without showing our Optin. Callback will send feedback on whether the permissions were granted.

```
public static void requestPermissions(@NonNull Activity mActivity, ArrayList<String>
permissions, boolean includeCdoPermissions, CalldoradoFullCallback overlayCallback)
```

Method which can be used to request Overlay permission to be used by publishers not showing our Optin and hence not using our permission handling. Callback will send feedback on whether the permission was granted. In the callback use runOnUiThread(new Runnable..) in the callback to continue flow:

```
public static void requestOverlayPermission(@NonNull Activity mActivity,
CalldoradoOverlayCallback overlayCallback)
```

Method which may be used to show a Caller SDK Aftercall screen. Use it to provide phone number search functionality (EditText) for the users and generate additional revenue:

```
public static void search(@NonNull Activity context, @NonNull CDOPhoneNumber
cdoPhoneNumber)
```

Method which will show the Aftercall screen of the latest phone call. Use to generate more ad revenue:

```
public static void showLastCallScreen(Activity context)
```

One of several alternative Calldorado initialization methods, which contains a callback providing information on user feedback to Optin and permissions. Use for either flow control telling you when initialization has finished or for following up on user actions:

```
public static void startCalldorado(@NonNull Activity mActivity, NonNull
CalldoradoFullCallback initCallback)
```

3.0 Caller SDK dependencies

Differing versions of dependencies used by both host app and Caller SDK may cause build errors and possibly crashes. Ways to get the dependencies synchronized is to either change the version of the dependency in the host app or to exclude the dependency provided by Caller SDK as follows:

```
implementation ('com.calldorado:calldorado-release:6.0.x.xxxx@aar ') {
    transitive = true
    exclude group: 'com.android.support', module: 'appcompat-v7'
}
```

Beware that running different versions of Caller SDK dependencies than what we have tested may lead to undesirable behavior.

To ensure that your dependencies gets updated properly when upgrading CDO, it is good practice to ‘clean --refresh-dependencies’ after changing the version in the Gradle file.

Feature requiring	Dependency	CDO 5.9	CDO 6.0
CDO core	com.squareup.picasso:picasso	2.5.2	2.71828
CDO core	com.android.support.constraint:constraint-layout	1.1.3	1.1.3
CDO core	androidx.cardview:cardview		1.0.0
CDO core	android.appcompat:appcompat		1.1.0
CDO core	Com.google.android.material:material		1.1.0
CDO core	com.android.volley:volley	1.1.0	1.1.1
CDO core	com.android.installreferrer:installreferrer		1.1.2
CDO core	Androidx.recyclerview:recyclerview		1.1.0
CDO core	Androidx.annotation:annotation		1.1.0
CDO core	Androidx.legacy:legacy-supoort-v4		1.0.0
CDO core	Androidx.work:work-runtime		2.3.4
CDO core	Androidx.transition:transition		1.3.1
CDO core	com.android.installreferrer:installreferrer	1.0	1.1.2
CDO core	com.luckycatlabs:SunriseSunsetCalculator	1.2	1.2
Google ads	com.google.android.gms:play-services-ads	17.1.1	18.3.0
Facebook Ads	com.facebook.android:audience-network-sdk	5.6.0	5.6.1
Mopub ads	com.mopub:mopub-sdk-banner	5.4.1	5.4.1
Mopub ads	com.mopub:mopub-sdk-native-video	5.4.1	5.4.1
Smaato ads	com.smaato:soma	9.1.3	9.1.3
CDO core	com.google.android.gms:play-services-awareness	16.0.0	16.0.0
CDO core	com.google.android.gms:play-services-location	16.0.0	16.0.0

Table 2 Dependencies for recent Caller SDK versions

4.0 Caller SDK Handling of 2018 GDPR Rules for EEA Countries

To make Caller SDK compliant with the GDPR rules of the EEA countries, we have made following changes to Caller SDK:

1. During the first initialization of Caller SDK, a load timer will be shown until the screen displays opt-in/permissions for EEA users.
2. We have removed the option to share information about a phone number if the number origin is an EEA country.
3. We have added two new entries to Caller SDK settings under the heading 'Data'. Here a user from EEA countries will get options to either enable/disable personalized ads or to erase memory of the app.

Activating the GDPR rules in Caller SDK is the new default, but can be deactivated through the following entry in your AndroidManifest under the <Application> tag:

```
<meta-data
    android:name="com.calldorado.EEAMode"
    android:value="disable" />
```

4.1 Trigger Actions on User Resetting Personal Data

In relation to GDPR, a user has the rights to clear app history. Due to this requirement, users in EEA countries can select 'Delete App Data' from the Caller SDK settings menu. If the delete option is selected, the app will be reset like clearing app data through the device settings. To be fully GDPR compliant at data reset, we trigger the method below, which allows a host app to clear data stored in files or reset firebase data.

```
public static void onCdoDataReset(Context context) {
      //Example. Resetting firebase on CDO data reset
       FirebaseAnalytics mFirebaseAnalytics =
FirebaseAnalytics.getInstance(context);
       mFirebaseAnalytics.resetAnalyticsData();
}
```

The method above is triggered using reflection. The method must be placed in the launch activity of your app and it must have the exact same name or reflection will throw an exception.

5.0 Caller SDK User Opt-In (Calldorado 5.3+)

Please note that by default you need to prvide your own optin and you will have to notify Caller SDK about user consent. If you do not turn it on you will need to get user consent for EULA and Privacy Policy that conforms to our guidelines from the users of your app, contact your key account manager if you are in doubt of what to include here, and then notify Caller SDK when these are obtained. It is mandatory to inform Caller SDK about user consent, otherwise the SDK will show a popup on the Aftercall.

```
Calldorado.acceptConditions(this, conditionsMap);
```

Where *this* is the current context and *conditionsMap* map is a HashMap of the following format:

```
HashMap<Calldorado.Condition, Boolean> conditionsMap = new HashMap<>();
conditionsMap.put(Calldorado.Condition.EULA, true);
conditionsMap.put(Calldorado.Condition.PRIVACY_POLICY, true);
```

Note that you can get the current acceptance values by calling the following method:

```
Calldorado.getAcceptedConditions(this);
```

Which returns a HashMap of the following format:

```
Map<Calldorado.Condition, Boolean>
```

5.1 How to handle permissions and opt-in

For Caller SDK versions lower than 5.3, permissions required by Caller SDK were handled separately from the Optin, but with 5.3, permissions have been integrated into the Optin and currently won't be handled if the Optin is not shown.

This guide shows how permissions can be handled locally before Caller SDK is initialized.

Handling permissions and startCalldorado()

For regular permissions, you may already have your own way of handling those and so could include Caller SDK permissions into your current code, but below is a basic example of a full permission request leading to a Caller SDK initialization.

Regular permissions in a stable Activity:

```
private int permissionRequestCode = 356;
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ArrayList<String> permissionList = new ArrayList<>();
    //Essential for Calldorado to work -One request (Phone)
    permissionList.add(Manifest.permission.READ_PHONE_STATE);
    //Optimal (Phone)
    permissionList.add(Manifest.permission.CALL_PHONE);
    permissionList.add(Manifest.permission.ANSWER_PHONE_CALLS);

    //Optimal -better user experience -One request (Contacts)
    permissionList.add(Manifest.permission.WRITE_CONTACTS);
    permissionList.add(Manifest.permission.READ_CONTACTS);
    //Optimal -ads can use location to serve more fitting ads -One request
    permissionList.add(Manifest.permission.ACCESS_COARSE_LOCATION);
    permissionList.add(Manifest.permission.ACCESS_FINE_LOCATION);

    //Optimal -Needed to read phone number on Pie+ devices. Requires permission
from Google to use from early 2019
    //permissionList.add(Manifest.permission.READ_CALL_LOG);
    ActivityCompat.requestPermissions(this, permissionList.toArray(new
String[permissionList.size()]), permissionRequestCode);
}
```

Listen for feedback and only request Overlay permission if phone permission has been granted.

Overlay permission is special in that it is granted automatically to some apps downloaded from the Play store for most devices, but there are devices for which it is not automatically granted and so it should be handled locally to catch those users. We have added a public method to handle the Overlay permission, which includes a Boolean callback with the user reaction to the request. The method can be seen used in below code snippet.

```
/**
 * listen for feedback from default permission requests
 */
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[]
permissions, int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    if (grantResults == null || grantResults.length == 0) {
        return;
    }
    Log.d(TAG, "onRequestPermissionResult.  requestCode = " + requestCode + ",
permissions = " + Arrays.toString(permissions) + ",        grantResults = " +
Arrays.toString(grantResults));
    if (requestCode == permissionRequestCode) {
        for (int i = 0; i < grantResults.length; i++) {
            if (grantResults[i] == PackageManager.PERMISSION_GRANTED) {
                switch (permissions[i]) {
                    case "android.permission.READ_PHONE_STATE":
                        //No reason to init Calldorado or request Overlay
permission unless Phone permission has been granted
                        Calldorado.requestOverlayPermission(this, new
Calldorado.CalldoradoOverlayCallback() {
                            @Override
                            public void onPermissionFeedback(boolean
overlayIsGranted) {
                                Log.d(TAG, "onPermissionFeedback()
overlayIsGranted = " + overlayIsGranted);
                                Calldorado.startCalldorado(MainActivity.this);
                            }
                        });
                        break;
                    default:
                        break;
                }
            } else if (grantResults[i] == PackageManager.PERMISSION_DENIED) {//
                switch (permissions[i]) {
                    default:
                        break;
                }
            }
        }
    }
}
```

Caller SDK can be initialized from the callback to the Overlay permission, like we have done above. It can be initialized only if the Overlay permission has been granted or even separately from (before or after) the Overlay permission request. We do recommend requesting the Overlay permission both to achieve a better user experience and due to showing an Aftercall screen without the call info overlay could result in a Google policy violation.

6.0 Using Your Own Opt-In

The best solution for onboarding users, is to create your own onboarding flow/process (OPT-IN). A good solution focuses on explaining the user what the Caller SDK does in the context of the app. Please also consider how and when any permissions are asked.

To ensure a good Caller SDK integration you need to follow the below steps. After the steps you will be taken through each of the steps in details with code examples:

1. Make sure that the Caller SDK opt-in logic is disabled *(off by default - only needed if you have used the Caller SDK opt-in in past releases of your app)*
2. Check if the user has already opt'ed in *(You should always check for this but if you have been using the Caller SDK opt-in in the past this is an important step since you do not want to show your own opt-in to users that have accepted the Caller SDK's opt-in in the past)*
3. Get consent from the user (using your own UI) for:
    a. The end user license agreement (EULA)
    b. The privacy policy
4. Inform Caller SDK about consents given by user
5. Start the Caller SDK
6. Request the permissions needed by the Caller SDK:
    a. Manifest.permission.*READ_PHONE_STATE*
    b. Manifest.permission.*CALL_PHONE*
    c. Manifest.permission.*ANSWER_PHONE_CALLS*
    d. Manifest.permission.*WRITE_CONTACTS*
    e. Manifest.permission.*READ_CONTACTS*
    f. Manifest.permission.*ACCESS_COARSE_LOCATION*
    g. Manifest.permission.*ACCESS_FINE_LOCATION*
    h. Manifest.permission.*READ_CALL_LOG* – Note that you will need to have special approval from Google in order to use this permission, ***if you do not have approval*** do not request it. **IMPORTANT!** Please make sure ***you have read section 1.2*** above about the call log permission before you proceed here
    i. android.permission.SYSTEM_ALERT_WINDOW, note that the request of this permission follows a different pattern than the above permissions. This should be the last thing to do in the onboarding process.
7. Make sure that you start the Caller SDK once every time your app is launched

You can use the above list as a todo-list to make sure you have gone through all the needed steps. In the following we will go into details about how to complete each of these steps from a programmatic point of view.

Step 1)

You can disable the Caller SDK opt-in by removing the following from your manifest (or validating that it is not there):

```
<meta-data
    android:name="com.calldorado.LocalOptin"
    android:value="enable"/>
```

Step 2, 3, 4, 5 and 6)

The remaining steps can be completed by doing the following in your code:

```
private int permissionRequestCode = 356;

// Step 2:
Map<Calldorado.Condition, Boolean> cdoConditions =
Calldorado.getAcceptedConditions(this);
if (cdoConditions.containsKey(Calldorado.Condition.EULA)) {
    if (cdoConditions.get(Calldorado.Condition.EULA)) {
        // TODO User has already accepted conditions. No need for you to show Optin
        onBoardingSuccess();
    } else {
        // Step 3
        // TODO show your means of boarding the user
        if(success){
            // Step 4
            HashMap<Calldorado.Condition, Boolean> conditionsMap = new HashMap<>();
            conditionsMap.put(Calldorado.Condition.EULA, true);
            conditionsMap.put(Calldorado.Condition.PRIVACY_POLICY, true);
            Calldorado.acceptConditions(this, conditionsMap);
            // Step 5
            onBoardingSuccess();
        }
    }
}

private void onBoardingSuccess() {
    // Step 5 (continued)
    Calldorado.startCalldorado(this);
    // Step 6
    requestCdoPermissions();
}

private void requestCdoPermissions() {
    ArrayList<String> permissionList = new ArrayList<>();
    permissionList.add(Manifest.permission.READ_PHONE_STATE);
    permissionList.add(Manifest.permission.CALL_PHONE);
    permissionList.add(Manifest.permission.ANSWER_PHONE_CALLS);
    permissionList.add(Manifest.permission.WRITE_CONTACTS);
    permissionList.add(Manifest.permission.READ_CONTACTS);
    permissionList.add(Manifest.permission.ACCESS_COARSE_LOCATION);
    permissionList.add(Manifest.permission.ACCESS_FINE_LOCATION);
    permissionList.add(Manifest.permission.READ_CALL_LOG); //TODO this permission needs
permission from Google to use (no permission -> exclude)
    ActivityCompat.requestPermissions(this, permissionList.toArray(new
String[permissionList.size()]), permissionRequestCode);
}
```

Step 7) Requesting Overlay permission

To request the system overlay permission you can use the following code to request the overlay permission once the request for the above permissions has been completed:

```
Calldorado.requestOverlayPermission(this, new Calldorado.CalldoradoOverlayCallback() {
    @Override
    public void onPermissionFeedback(boolean overlayIsGranted) {
```

```
        //Log.d(TAG, "onPermissionFeedback(): overlayIsGranted = " + overlayIsGranted);
    }
});
```

NB! Note that you should run all of the above code in a non-temporary Activity, for instance the activity where you opt-in your users or the main Activity of your app. Running the above code in a splash activity that serves as a loading screen or similar is not recommended and thus not guaranteed to produce the results you need given the brevity of the life-cycle of these kind of activities.

Final Step)

Once you have informed the Caller SDK about the user's consent, started the Caller SDK, and requested the permissions from the user, the Caller SDK has everything it needs to run. From this point on it is important that you start the Caller SDK at least once every time your app is launched. Actually, it is okay to do this before you opt-in the user. You can do so by calling the following in the first non-temporary activity the user sees when the user is using your app normally (for instance in the activity's on-create method):

```
    Calldorado.startCalldorado(this);
```

7.0 Handling premium users

In order to show a premium ad-free experience to the users, you need to add your Google Play Console SKU's to my.calldorado.com.

Before Calldorado 5.9.19

When your user has finalized the purchase, please call this code in Calldorado:

```
Intent premiumIntent = new Intent(this,
com.calldorado.android.actionreceiver.ActionReceiver.class);
premiumIntent.setAction("com.calldorado.android.intent.PREMIUMUPDATE");
this.sendBroadcast(premiumIntent);
```

Calldorado 5.9.19 and up

When your user has finalized the purchase, please call this one-liner in Calldorado:

```
Calldorado.updatePremiumUsers(this);
```

8.0 Known issues

If you find the error message "Android resource linking failed" while trying to generate an apk, add the following code to the app gradle.

```
android {
    ...
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
}
```

If you wish to disable Google's auto backup for your app, please add the following to the application tag of your app's manifest (Not needed as of CDO version 5.9.18):

```
android:allowBackup="false"
tools:replace="android:allowBackup"
```

9.0 Version

Version	Date	Description	Resp.
1.0.0	16/01-2019	First release	MOH
1.0.1	23/01-2019	Updates to APIs	JOB
1.0.2	28/01-2019	Added table with environment settings	MOH
1.0.3	31/1-2019	Suggestion comments added	PB
1.0.4	6/2-2019	Spelling proofing and correction of the existing document. Replaced "Calldorado"-name with "Caller SDK"-name (except for code ofcourse) Fixed the known issue: "6.1 App Icon and Name Not Showing Properly in CDO Opt in" in develop (coming from CDO 5.8 and onwards)	BJ, PB
1.0.5	12/2-2019	Response to suggestions leading to minor formatting and wording changes.	MOH
1.0.6	18/2-2019	Merged customization guide into doc	MOH
1.0.7	21/2-2019	Updates regarding CDO 5.8+ integration	JOB
1.0.8	29/4-2019	Updates regarding CDO 5.9+ integration	JOB
1.0.9	23/5-2019	Updated the integration with instructions for custom opt-in integrations.	SSH
1.11	06/06-2019	Removed snapshots from gradle dependencies and updated implementation of CDO to be without snapshots. Snapshots are for internal use only. Added chapter for handling premium users.	PB
1.15	11/03-2020	Updates regarding integration for TargetSdkVersion, generalization of CDO version number, dependency versions	SIP
1.16	15/09-2020	Updates regarding integration for TargetSdkVersion, generalization of CDK Version number, CompileSdkVersion	SIP
1.17	17/11-2020	Updated the section 1.4.2	GSK